UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Métodos de entrenamiento de redes neuronales basados en inyección de ruido.

Juan Alvear Alonso-Villaverde
Tutor: Alberto Suárez González

JUNIO 2018

Métodos de entrenamiento de redes neuronales basados en inyección de ruido.

AUTOR: Juan Alvear Alonso-Villaverde
TUTOR: Alberto Suárez González

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2018

Resumen (Castellano)

El objetivo del aprendizaje automático supervisado es construir un sistema que permita predecir la clase de un ejemplo a partir de su descripción, generalmente un conjunto de atributos. Una de las dificultades que surgen cuando se realiza este proceso de inducción es el sobreaprendizaje. Se dice que un sistema sobreaprende cuando las mejoras en la calidad de sus predicciones para los ejemplos que se han utilizado para su entrenamiento, se traducen en un empeoramiento de sus tasas de acierto en el problema; en concreto, en ejemplos independientes de los utilizados para entrenamiento.

Algunos de los posibles y más extendidos enfoques para abordar este problema son: La regularización mediante penalizaciones, la regularización mediante parada temprana en el aprendizaje (*early stopping)*, o en redes profundas, mediante la técnica conocida como *dropout*.

Un enfoque también común es la inyección de ruido en la red neuronal artificial, ya sea en la capa de entrada, en la capa oculta o en la capa de salida.

Este Trabajo de Fin de Grado estudia este último enfoque, es decir, inyección de ruido en la capa de salida de la red neuronal artificial. Se realiza un estudio sobre el impacto de inyectar ruido en los resultados de clasificación.

El ruido se añade únicamente en la fase de entrenamiento, mediante la inclusión de una neurona generadora de ruido (normal o uniforme) en la capa oculta. Afectando por lo tanto a la salida de la red neuronal artificial.

Para la realización de este estudio se ha diseñado e implementado una red neuronal artificial que permite la introducción de ruido escalable en la fase de entrenamiento. Asimismo se han diseñado un conjunto de test que permiten realizar las pruebas correspondientes.

La mayoría de estas pruebas han sido realizadas sobre conjuntos de datos correspondientes a problemas de clasificación binaria reales.

Abstract (English)

The target of the supervised machine learning is to build a system that is able to predict the class of an example, generally a set of attributes. One of the difficulties that arises when this process is performed is the overlearning. It is said that a system overlearns when the improvement in the quality of its predictions for the examples that has been used for its training, are translated in a deterioration of its success ratios; specifically, in independent examples of those used for training.

Some of the possible and more extended approaches to resolving this problem are: regularization through penalties, regularization through early stopping in the training phase or, in deep neural networks, through dropout.

Another approach is the injection of noise in the artificial neural network, whether in the input layer, the hidden layer, or in the output layer.

This Bachelor Thesis studies the last approach; the injection of noise in the output layer of the artificial neural network. A study is carried out on the impact in the classification results.

Noise is injected only in the training phase, with the addition of a noise generator neuron (normal or uniform) in the hidden layer and, therefore, affecting the neural network output.

In order to carry out this study, an artificial neural network has been designed and implemented, which allows the addition of scalable noise in the training phase. Likewise it has been designed a set of tools that allow us to perform the relevant tests.

These tests have been executed, in their majority, over real binary classification problem datasets.

Palabras clave (Castellano)

Ruido, redes neuronales, perceptrón multicapa, aprendizaje automático, clasificación automática, sobreaprendizaje, sobreentrenamiento.

Keywords (English)

Noise, neural networks, multilayer perceptron, machine learning, automatic classification, overlearning, overtraining.

Agradecimientos

A mi familia, en especial a mi hermano y mis padres, y a todas las personas que han hecho que este viaje, además de posible, haya sido una gran experiencia.

Gracias.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

Este TFG se basa en la idea de estudiar el comportamiento de una red neuronal cuando se añade ruido a su fase de entrenamiento en la capa de salida. En concreto, la arquitectura considerada incluye una neurona generadora de ruido en la capa oculta.

Este es un enfoque poco común para la resolución del problema del sobreaprendizaje. El objetivo es comprobar si esta modificación aporta estabilidad a la red neuronal artificial, y si soluciona el problema del sobreentrenamiento, es decir, le permite generalizar mejor.

1.2 Objetivos

El objetivo principal de este TFG ha sido estudiar cómo afecta la inyección de ruido de dos tipos (normal y uniforme) a la fase de entrenamiento de un perceptrón multicapa en la capa de salida, comparando los resultados obtenidos en la fase posterior de clasificación, tanto con, como sin dicho ruido.

El estudio se ha llevado a cabo en dos etapas:

- Investigar de qué forma se puede añadir una neurona generadora de ruido en librerías ya implementadas de Python, como ScikitLearn o Keras. En caso de no ser posible, implementar un perceptrón multicapa que permita añadir una neurona generadora de ruido escalable en su entrenamiento en la capa oculta.

- Una vez que se disponga de las herramientas necesarias para realizar la comparación, hacer pruebas sobre datasets de clasificación binarias de problemas reales.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

1. **Introducción:** Motivación, objetivos y organización de la memoria de este TFG.

2. **Estado del arte:** Se introduce el tema y se contextualiza, se explican nociones básicas de redes neuronales artificiales, se explica de forma conceptual el esquema tradicional de perceptrón multicapa con una capa oculta y el propuesto e implementado para la realización del TFG.

3. **Diseño:** Se resume la investigación previa a la decisión de diseñar e implementar el MLP con posibilidad de añadir ruido por cuenta propia. Se detalla la propuesta de MLP y se explica el dataset creado para las primeras pruebas.

4. **Desarrollo:** Se explica detalladamente qué librerías externas se han utilizado, cómo se ha implementado el MLP propuesto, el algoritmo de retropropagación, los métodos generados, cómo se ha implementado el *early stopping*.

5. **Pruebas y resultados:** Se explica cómo se ha comprobado que la implementación del MLP era correcta y el procedimiento seguido para realizar el estudio de cómo afecta el ruido en la fase de entrenamiento de una red neuronal con distintos datasets y distintas condiciones. Se muestran los resultados de las pruebas sobre cada dataset.

6. **Conclusiones y trabajo futuro:** Se exponen las conclusiones globales del estudio realizado y se proponen aspectos a ampliar respecto al TFG.

2 Estado del arte

2.1 El aprendizaje automático

El aprendizaje automático es una rama de la Inteligencia Artificial que trata el diseño de sistemas capaces de aprender a partir de datos El objetivo es identificar relaciones en los datos utilizados para construir el sistema que generalicen bien. Decimos que un sistema generaliza cuando las relaciones identificadas son también relevantes para caracterizar otros ejemplos, independientes de los utilizados en la construcción del sistema. .

Al igual que la Inteligencia Artificial el aprendizaje automático está en auge, probablemente debido al gran número de aplicaciones que tiene, por ejemplo en la medicina (Detección de enfermedades, estudio del ADN), en el reconocimiento de patrones (Detección de fraudes), motores de búsqueda, publicidad, entre otros.

Entre los temas del aprendizaje automático se encuentran las redes neuronales artificiales.

2.2 Redes neuronales artificiales (RNA)

Las redes neuronales artificiales o RNA están inspiradas en las redes neuronales biológicas, como las que conforman nuestro cerebro. Están formadas por neuronas estructuradas en capas. Las neuronas de distintas capas están conectadas. Cada neurona recibe señales de otras neuronas, moduladas por pesos sinápticos, los cuales pueden ser excitatorios o inhibitorios. La neurona transforma la señal recibida de manera no-lineal y la envía a otras neuronas.

Aunque hay muchos tipos de redes neuronales artificiales el objetivo de este TFG se centra en los perceptrones multicapa con propagación hacia adelante (*feed-forward Multilayer Perceptron*). En este tipo de redes neuronales las neuronas de una capa reciben señales de las de la capa anterior y, tras realizar el procesamiento correspondiente, las envían a neuronas de la capa siguiente

2.2.1 Perceptrón simple

Un perceptrón es una RNA formada por capas de neuronas conectadas entre sí, en el caso del perceptrón simple únicamente una capa de entrada y una de salida.

El problema de los perceptrones simples es que solo permiten resolver problemas de clasificación lineales *[1]*.

2.2.2 Perceptrón multicapa

Un perceptrón multicapa o MLP (por sus siglas en inglés MultiLayer Perceptrón) es aquel formado por varias capas de neuronas, en concreto una capa de entrada, una capa de salida, y como mínimo una capa intermedia a la que denominamos capa oculta.

Como se ve en la *Figura 1*, todas las capas a excepción de la de salida poseen una neurona adicional con valor de 1, a esta neurona corresponde a un término constante, de sesgo (*bias*). Sirve para ajustar el desplazamiento de la función de activación de la neurona a la que está conectada.

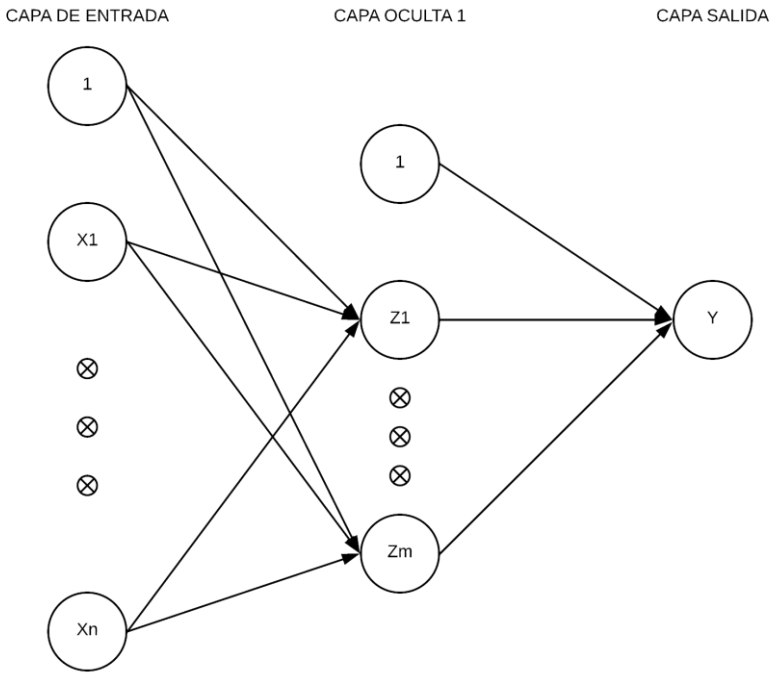

Figura 1. Esquema de perceptrón multicapa con 1 capa oculta.

La primera capa es la de entrada, con N + 1 neuronas. Cada neurona en esta capa representa un atributo o propiedad del ejemplo a clasificar. La predicción se realiza a partir de los valores del conjunto de atributos que describe el ejemplo.

La primera, y en este caso, única capa oculta tiene M + 1 neuronas, donde M es un valor que se elige manualmente, o tras haber realizado un estudio comparativo de la calidad de las predicciones de la red distintos valores de dicho parámetro (por ejemplo mediante validación cruzada).

El número de capas ocultas también es una elección en el diseño de la red. No obstante hay que tener en cuenta que añadir muchas capas ocultas, puede no

beneficiar al aprendizaje y aumentar el coste computacional, resultando en un sistema innecesariamente más lento. En un problema de tamaño pequeño como los utilizados en este trabajo no es necesario aumentar el número de capas ocultas, sin embargo, si el tamaño de los datos a tratar es mayor, por ejemplo, para tratar imágenes, sí que está demostrado que aumentar M es necesario, ya que se obtienen mejores resultados*[7]*.

Todas las neuronas están conectadas con las neuronas de la siguiente capa a excepción de la neurona *bias* de dicha capa. Por lo tanto la función de activación de una neurona depende de la entrada que reciba, es decir de todas las neuronas de la capa anterior.

Por lo descrito anteriormente, las neuronas *bias* siempre están activadas con un valor de 1, que será regulado por el peso de sus enlaces con el resto de neuronas a las que están conectadas.

El funcionamiento de un MLP se divide en dos fases:

1. Fase de entrenamiento: La red neuronal recibe un conjunto de datos como entrada y se entrena mediante la observación de esos datos, hay distintos métodos de entrenamiento, en el caso de este TFG se ha utilizado el algoritmo *backpropagation* o retropropagación mediante descenso de gradiente que se explica en detalle en el *Apartado 4*. Conceptualmente, y ya que por el conjunto de datos de entrenamiento sabemos cual es el valor esperado, se basa en la idea de medir el error a la salida, y ajustar los pesos desde la capa de salida hasta la de entrada para ir compensando ese error.

2. Fase de predicción: Una vez entrenada la red ya pueden clasificarse vectores de atributos de forma sencilla y eficaz.

El entrenamiento del perceptrón multicapa se basa en ajustar los pesos de todos los enlaces entre neuronas mediante diferentes métodos, el que se utiliza en este TFG es el descenso por gradiente tanto *Online* como *Batch* que se explicará más detalladamente en posteriores apartados.

2.2.3 Perceptrón multicapa propuesto

Para la realización de este TFG se ha diseñado e implementado un MLP con una única capa oculta y una única neurona en la capa de salida. Es decir, el objetivo de nuestras pruebas son los problemas binarios ya que la salida de nuestra red neuronal es binaria.

A esta capa oculta se le agrega de forma opcional una neurona generadora de ruido normal o uniforme, este ruido puede ser escalado mediante otro parámetro. Es importante escalar este ruido correctamente, como se explica en el correspondiente apartado.

2.3 Trabajo anterior

El problema de sobreaprendizaje, ya mencionado anteriormente, es uno de los centrales en el aprendizaje automático. Los métodos que han sido utilizados para solventar este problema son: Regularización mediante penalizaciones (por ejemplo, la inclusión de un término L2 en la función de pérdida); mediante *early stopping*, es decir, monitorizado la evolución del error en una partición de validación en función del número de épocas de aprendizaje y parando cuando este se deteriore; o, en el caso de las redes neuronales profundas, mediante *dropout*, que de forma resumida, consiste en ignorar de forma aleatoria conjuntos de neuronas.

Se han realizado algunos estudios sobre el uso del ruido como forma regulatoria del entrenamiento, aunque este enfoque es mucho más inusual:

- Introducción de ruido en la capa de entrada *[2]*: Donde se defiende que debido al algoritmo de entrenamiento utilizado (retropropagación), y a que un mismo ejemplo se ve ligeramente modificado por el ruido entre épocas manteniendo su clase intacta, mejora la estabilidad de la red neuronal y por lo tanto los resultados obtenidos.

- Introducción de ruido en la capa oculta *[9]*: Se introduce ruido en las neuronas de la capa oculta, esto afecta de dos formas a la red neuronal: Aprendizaje del modelo y regularización, a mayor ruido será más difícil para red neuronal aprender el modelo correctamente pero será más estable al ruido (regularización), es por esto que debemos encontrar un punto de equilibrio que beneficie a ambos elementos, de aquí la importancia de utilizar ruido escalado. La regularización mediante *dropout* puede entenderse como un caso muy específico de este método, introduciendo el ruido como multiplicadores de valor 0, es decir, ignorando el peso de algunas neuronas aleatoriamente.

Sin embargo, el objetivo de este TFG es la inyección de ruido en la salida y el estudio de los resultados del mismo. Aunque anteriormente se ha estudiado la inyección de ruido en la capa de salida, ha sido en otros términos *[3]*: Velocidad de convergencia y cómo ayuda esto a escapar de los mínimos locales, problema conocido también del descenso por gradiente.

3 Diseño

3.1 Estudio previo

Antes de diseñar e implementar se contempló como posibilidad utilizar librerías existentes que permitieran añadir una neurona generadora de ruido en una capa oculta, con el fin de poder dedicar un mayor esfuerzo a la realización de pruebas y al propio estudio.

En concreto:

- ScikitLearn*[5]*: Aunque su uso es muy extendido y permite distintos ajustes, no permite configurar las capas ocultas por lo que no sirve para realizar el estudio.

- Keras*[6]*: Permite la creación de redes neuronales por capas pero no por neuronas por lo que seguía sin permitir la adición de una neurona generadora de ruido, sin embargo sí que permite introducir capas de ruido pero ese no es el objetivo de este TFG.

Tras realizar ambos estudios, se decidió implementar el MLP que permitiera añadir 1 neurona de ruido por cuenta propia, como se detalla en el apartado 3.3.

3.2 Dataset de pruebas

Para poder realizar pruebas sobre la implementación del MLP propio tanto con como sin ruido, y antes de enfrentarlo a un problema de clasificación binario real, se decidió crear un dataset no tan complicado pero tampoco tan simple.

El dataset creado está formado por 2 clases, cuyos componentes son puntos del plano representados por sus coordenadas X e Y. La generación de los puntos de cada clase era dada por la probabilidad de dos distribuciones normales o gaussianas.

3.2.1 Distribución normal

Es una distribución probabilística cuyo dominio va desde -∞ hasta +∞. De forma vulgar puede afirmarse que tiene forma de campana y una de sus propiedades más importantes es que el área que encierra dicha campana desde -∞ a +∞ es 1.

La función de densidad de la distribución normal unidimensional viene definida por la expresión:

$$N(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \quad -\frac{(x-\mu)^2}{2\sigma^2}$$

Se define por dos parámetros:

- La media μ donde su valor es máximo, y está ubicado su centro de simetría horizontal.
- La desviación típica σ que indica la distancia a la que se encuentran ambos puntos de inflexión de la curvatura.

A continuación se muestran varios ejemplos:

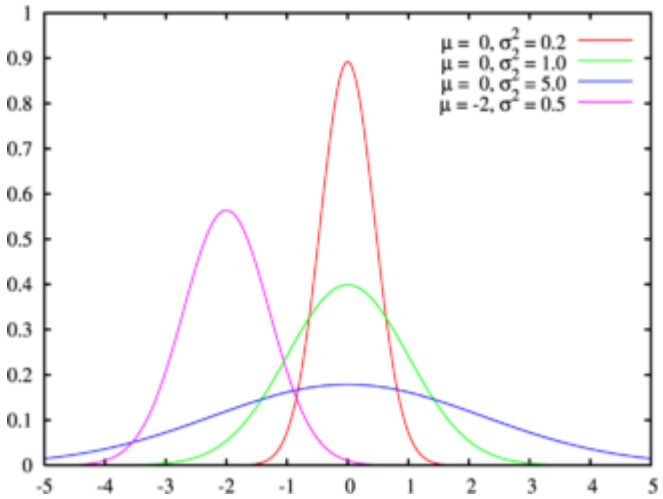

Figura 2. Ejemplo de distribuciones gaussianas dados distintos parámetros. Fuente: Wikipedia.

La distribución normal estándar es aquella con media 0 y varianza 1, representada en la *Figura 2* mediante la línea verde.

Un ejemplo de distribución normal bidimensional, donde el color y la altura de cada punto representa la probabilidad de que un punto elegido al azar siguiendo esa distribución sea ese.

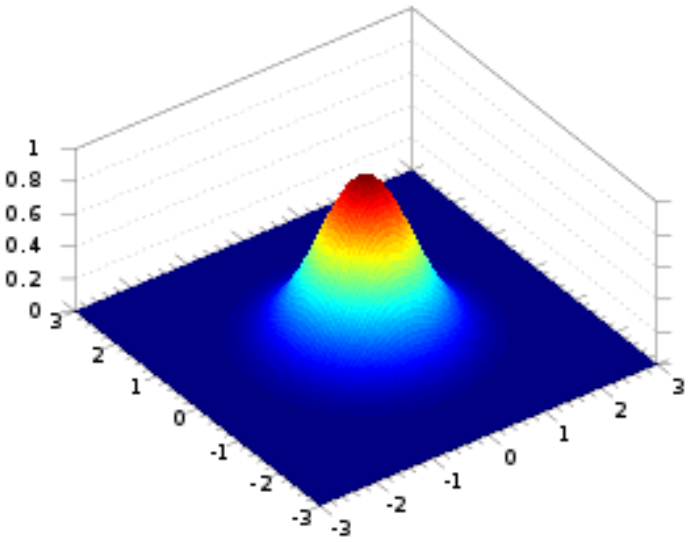

Figura 3. Distribución normal bidimensional con centro en el origen de coordenadas, esto es μ = (0, 0) y matriz de covarianzas igual a la matriz unidad.

La función de densidad extendida a D dimensiones, donde μ_D es el vector de medias yΣ_{DD}es la matriz cuadrada de covarianzas de la distribución. Σ_{ij}tiene como elementos de la diagonal principal las varianzas de cada dimensión, y el resto de los elementos las covarianzas entre las correspondientes dimensiones i y j, por ejemplo, Σ_{11}es la varianza de la dimensión 1, Σ_{12} es la covarianza entre las dimensiones 1 y 2 que es equivalente a Σ_{21}

Además $|v|$ denota el módulo de v yv^T el vector traspuesto de v.

$$[4]\ N(x_D \mid \mu_D, \Sigma_{DD}) = \frac{1}{\sqrt{|\Sigma|}\ (2\pi)^{\frac{D}{2}}} - \frac{(x-\mu)^T}{2}\Sigma^{-1}(x-\mu)$$

3.2.2 Dataset generado gaussianas

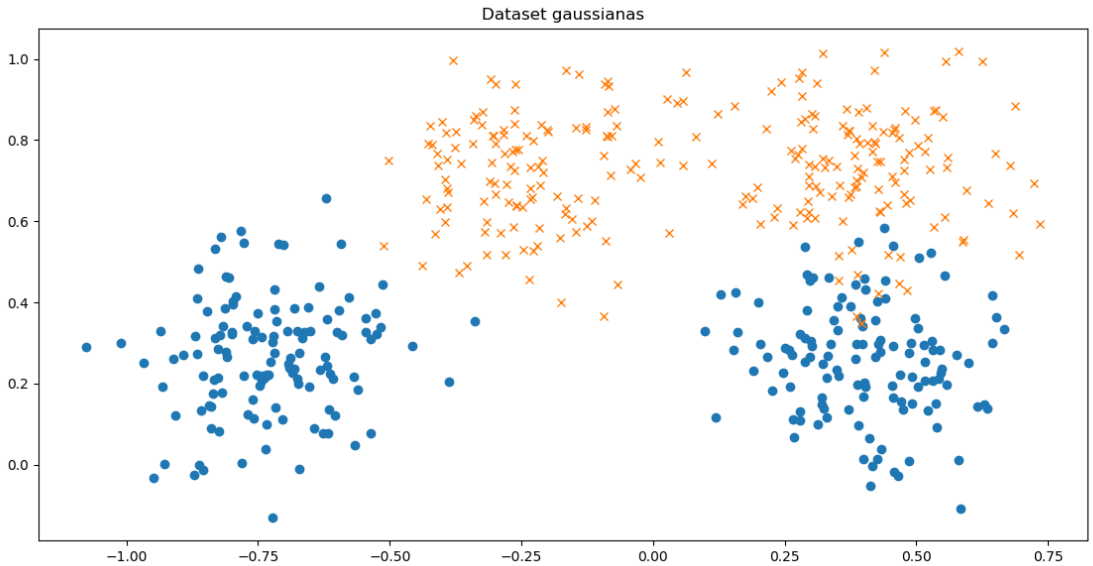

Figura 4. Dataset de gaussianas generadas con 500 puntos. Cada clase está formada por 250 individuos, es decir, es un dataset completamente equilibrado.

Como ya se ha explicado, cada una de las dos clases del dataset ha sido generado mediante la combinación de dos gaussianas. Idea extraída de *Referencia 10 [10]*:

- Clase 0: Con los centros de sus dos gaussianas definidos en las coordenadas (-0.7 , 0.3) y (0.4 , 0.25). Representada en la *Figura 4* mediante ‘o’.

- Clase 1: Con los centros de sus dos gaussianas definidos en las coordenadas (-0.25 , 0.7) y (0.4, 0.75). Representada en la *Figura 4* mediante ‘x’.

Las 4 gaussianas bidimensionales tienen como desviación típica 0.13. En total se han generado 500 puntos, 250 ejemplos de cada clase lo que se traduce en un dataset completamente equilibrado.

3.3 MLP con neurona generadora de ruido

3.3.1 Estructura

La estructura de este MLP, que permite la adición de una neurona con ruido escalable, sigue el esquema tradicional de un MLP de una capa, esto es, una capa de entrada, una capa oculta y una capa de salida (de una única neurona).

Los pesos entre la capa de entrada y la capa oculta están representados por una matriz V [N, M + 1] y los pesos entre la capa oculta y la capa de salida por un vector w [N + 1].

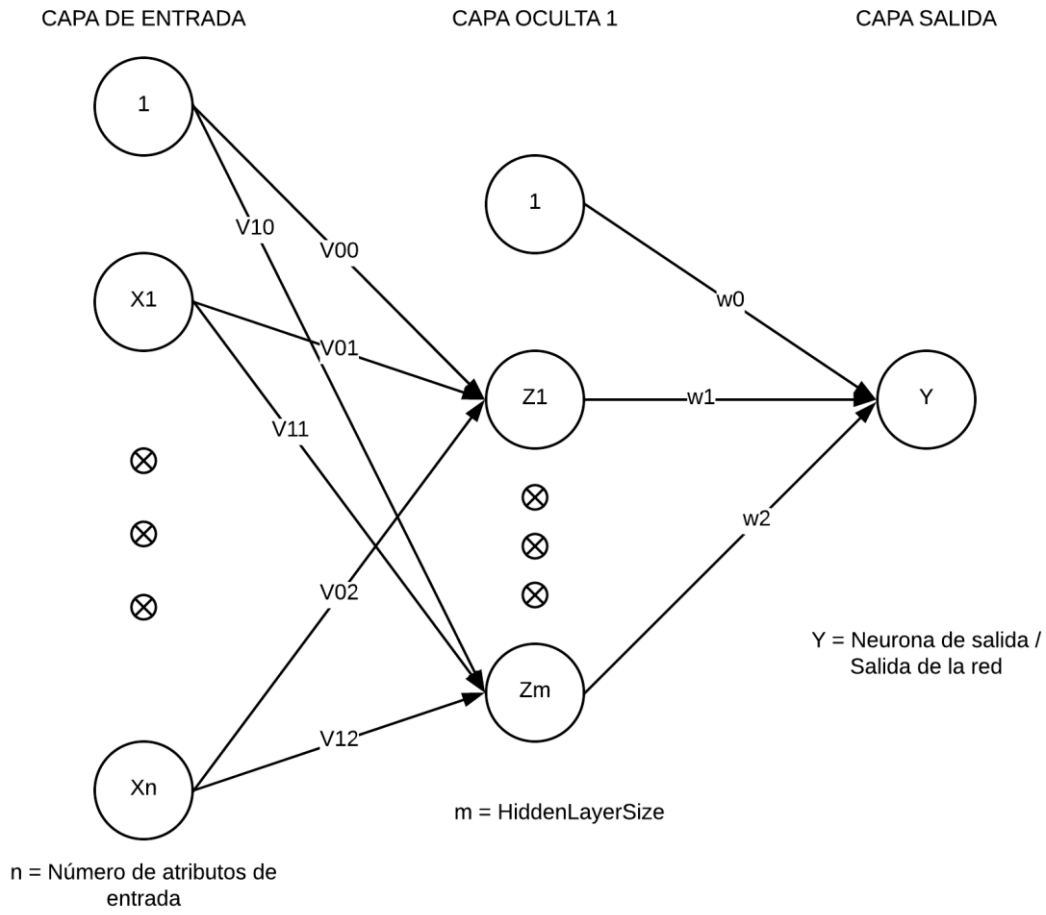

Figura 5. Esquema tradicional de un MLP de una capa.

Aunque conceptualmente hayamos añadido la neurona de ruido, y por las razones que se explican en los siguientes apartados, el peso de la conexión desde dicha neurona a la neurona de salida no está representado en ese vector, por lo tanto este mantiene las dimensiones N + 1 y no no N + 2.

Se ha decidido usar como función de activación de las neuronas la función sigmoidal que está definida como:

$$sigmoid(input) = \frac{1}{1 + exp^{-input}}$$

A este esquema hay que añadir la neurona generadora de ruido, cuyo peso no se entrena como los del resto de la red, es decir, es un nuevo parámetro “Escala del ruido” que deberemos de decidir a la hora de crear la red neuronal. Esto se ha hecho así ya que al ser su salida siempre aleatoria tendría problemas de convergencia y podría causar un mal funcionamiento del entrenamiento de la red neuronal artificial.

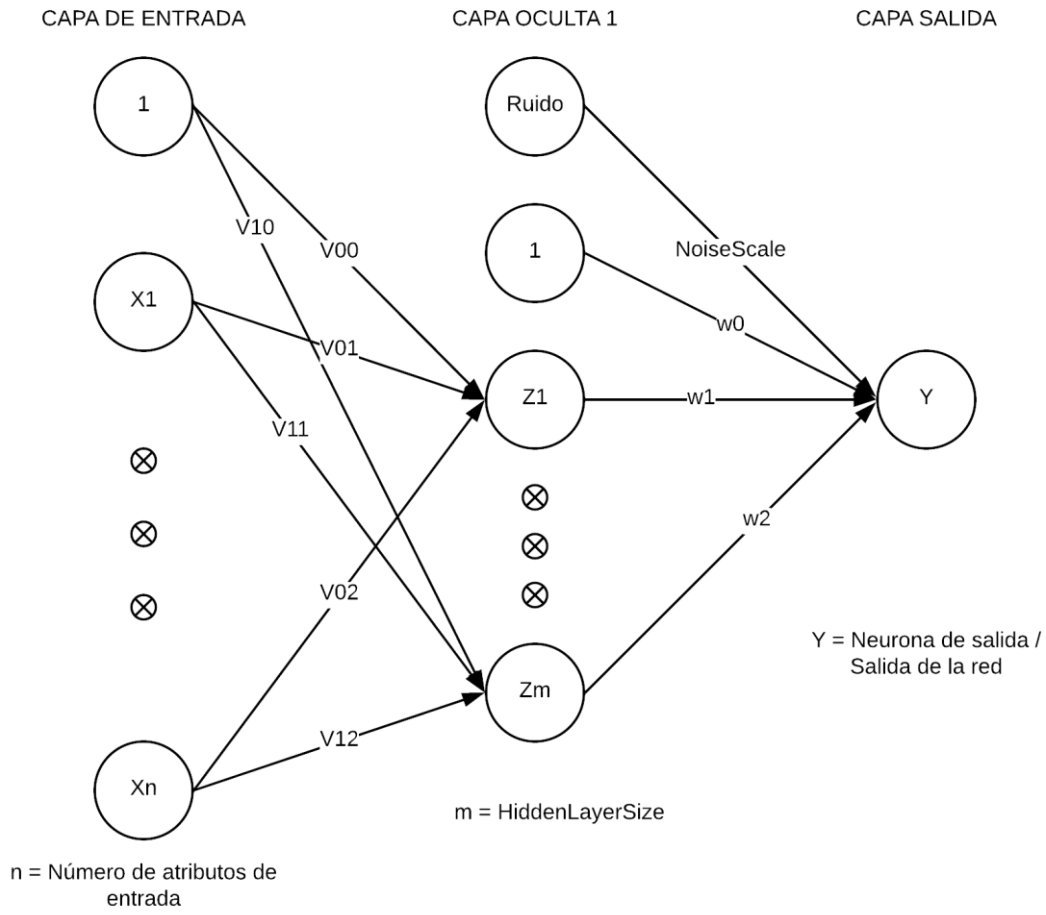

Figura 6. Esquema diseñado para un MLP de una capa con una neurona generadora de ruido. Nótese la nueva neurona encargada de generar el ruido respecto al esquema tradicional.

3.3.2 Parámetros de la red

Este nuevo esquema necesita de los siguientes parámetros que deben ser seleccionados manualmente.

- HiddenLayerSize: Es el número de neurona en la capa oculta. Heredado del MLP tradicional.

- LearningParameter: Es el desplazamiento de corrección/ajuste de los pesos en la época de aprendizaje, es un parámetro definido únicamente para la fase de entrenamiento de la red. Heredado del MLP tradicional.

- EpochNumber: Número máximo de épocas de la fase de entrenamiento, siempre y cuando la red neuronal no converja primero. Heredado del MLP tradicional.

- NoiseScale: Es la escala del ruido que genera la neurona.

Estos cuatro primeros parámetros son fundamentales para la realización de este TFG, por esta razón, aunque deban elegirse manualmente sus valores en las pruebas no han sido elegidos arbitrariamente, sino que como se explicará en el correspondiente apartado se han optimizado.

Adicionalmente a estos 3 parámetros principales, hay otros de menor relevancia pero que nos permiten personalizar más el funcionamiento de la red neuronal:

- NoiseType: Tipo de ruido generado por la neurona generadora de ruido.
- Tolerance: Tolerancia para el criterio de convergencia, es decir, para la detección de que la red neuronal ha convergido y no es necesario seguir entrenando. Se explica más en detalle en el apartado de *3.3.4 Early stopping*.
- UseBatch: Permite realizar el entrenamiento *Online* o por *Batch*.

3.3.3 La necesidad de la escala del ruido

Viene dada porque en función del problema que estemos afrontando y del número de neuronas de la capa oculta el valor que debe aportar el ruido puede ser muy distinto.

Un ejemplo, consideremos una red neuronal A con una capa oculta de 3 neuronas, y una red neuronal B con una capa oculta de 20 neuronas, situemos ahora en la neurona de salida, que debe aplicar el ruido a la suma de la salida de esas 3 o 20 neuronas, supongamos por un momento, que esas 3 o 20 neuronas están activas y su salida tiene valor 1, el valor de entrada de la neurona de salida será 3 + Ruido en la red neuronal A y 20 + Ruido en la red neuronal B.

De forma obvia, ambos problemas necesitan distintas escalas de ruido ya que si aplicaremos el mismo ruido a ambas redes neuronales su efecto sería muy distinto.

En este trabajo se ha utilizado tanto ruido uniforme en el dominio [-1, 1], como ruido normal estándar en el dominio $(-\infty , +\infty)$.

En principio, puede generarse el ruido de la forma que se desee, siempre y cuando se tenga en cuenta su dominio, y se utilice adecuadamente el parámetro de escala del ruido.

3.3.4 Early stopping

Es una técnica utilizada en el entrenamiento de las redes neuronales, para la detección de la convergencia del entrenamiento o cuando se está muy cerca de ella. Esto es útil para evitar el sobreentrenamiento.

Además de esta utilidad permite el ahorro de tiempo de ejecución ya que una vez detectado puede detenerse el entrenamiento y no es necesario llegar al máximo de épocas.

Lo que en algunos conjuntos de datos, innecesariamente grandes y con datos redundantes, se traducirá en un entrenamiento más eficaz y preciso.

4 Desarrollo

4.1 Librerías externas

Aunque la implementación del MLP se ha hecho desde cero por cuenta propia, se han utilizado dos librerías externas muy conocidas para realizar otro tipo de tareas:

- Numpy: Es una librería matemática de uso muy extendido. Se ha utilizado para el almacenamiento y uso tanto de matrices como de vectores. Así como otras funciones de utilidad como:
    - Cálculo de sumatorios de matrices D dimensionales.
    - Generadores de ruido, tanto de un único número como de matrices.
    - Operaciones con vectores como producto escalar.

- ScikitLearn: Se ha utilizado para la creación de particiones, más concretamente para la creación de particiones estratificadas para llevar a cabo el estudio de forma más estable. En un principio se utilizó como referencia para comparar con la propia implementación del MLP.

4.2 Implementación propia

Se ha creado la clase NoiseNeuralNetwork que recibe en su constructor el número de neuronas en la capa oculta, el parámetro de aprendizaje, y de forma opcional la escala del ruido a aplicar y la tolerancia para la detección de la convergencia.

Teniendo en cuenta que Número de neuronas en la capa oculta no incluye la neurona *bias*: Los pesos entre la capa de entrada y la capa oculta se almacenan como una matriz de numpy V[Número de neuronas en la capa oculta, Número de atributos + 1]. Con el fin de simplificar y dado que únicamente hay una capa oculta y una neurona de salida se han expresado los pesos entre dicha capa oculta y la salida de la red neuronal como un vector de numpy w[Número de neuronas en la capa oculta + 1]. Ambos + 1 se deben a las conexiones de las neuronas *bias* de ambas capas.

La función de activación que se ha utilizado es la sigmoidal que se definió en los apartados previos y transforma cualquier valor con dominio $(-\infty, +\infty)$ al dominio [0 , 1].

El método elegido para el entrenamiento y ajuste de los pesos de la red neuronal es la retropropagación tanto *Online* como *Batch*.

4.2.1 *Entrenamiento mediante retropropagación Online*

Algoritmo para entrenar la red neuronal, dado un conjunto de datos de entrenamiento:

1. Inicialización aleatoria de todos los pesos en el rango [-0.5, 0.5], tanto de la matriz de pesos como del vector.
2. Mientras *epoca_actual* < *numero_maximo_epocas* y no se detecte convergencia mediante Early Stopping.
    a. *epoca_actual* += 1
    b. Permutar todos los datos de entrenamiento.
    c. Por cada vector de atributos en el conjunto de atributos de entrenamiento:
        i. Calcular valores de las neuronas de la capa oculta. Sabiendo que el valor de cada una es la sigmoidal del producto escalar entre los atributos de entrada normalizados y los pesos de las neuronas de la capa de entrada conectadas a ella.
        ii. Calcular valor de la neurona de salida. Sabiendo que el valor es la sigmoidal del producto escalar entre los valores calculados de las neuronas de la capa oculta y los pesos entre estas y la neurona de salida. Aquí es cuando aplicamos el ruido escalable, añadiendolo al producto escalar, justo antes de aplicar la sigmoidal.
        iii. Comparar el valor esperado y el obtenido.
        iv. Calcular la corrección de pesos de la capa oculta teniendo en cuenta la comparación anterior, el parámetro de aprendizaje y el valor de las neuronas en la capa oculta.
        v. Calcular el error en la capa oculta teniendo en cuenta la comparación de valores de salida, los pesos de la capa oculta, los valores de las neuronas de la capa oculta.
        vi. Calcular la corrección de pesos en la capa de entrada, teniendo en cuenta el error calculado en la capa oculta y el parámetro de aprendizaje.
        vii. Aplicar ambas correcciones en los pesos, tanto en la capa oculta como en la de entrada.

    Observación: Como se explicó anteriormente, el factor de escala de ruido no cambia en ningún momento durante el entrenamiento.

La formulación matemática es la siguiente:

Denominamos V a la matriz de pesos [Número de neuronas en la capa oculta, Número de atributos + 1], w al vector de pesos [Número de neuronas en la capa oculta+1] y $\boldsymbol{x}$ al vector de entrada de atributos [Número de atributos], $y$el valor esperado de salida, Y el valor de salida de la red neuronal y ε el valor del ruido gaussiano o uniforme, ya escalado, en caso de haberlo.

Para calcular el valor de una neurona en la capa oculta, z_i representa la salida de la neurona i en la capa oculta, donde $i \in [0, hiddenLayerSize - 1]$ ya que el valor de la neurona *bias* es de 1 por lo que no hay que calcularlo:

$$z_i = sigmoid(V_i^T * x)$$

Para calcular el valor de la neurona de salida:

$$Y = sigmoid(z * w + NoiseScale * \varepsilon)$$

El error en la salida es:

$$\delta = (y - Y) * Y * (1 - Y)$$

El error en cada neurona de la capa oculta es, donde Δ_i es el error asociado a la neurona i de la capa oculta:

$$\Delta_i = z_i * (1 - z_i) * w_i * \delta$$

Por lo tanto la actualización de los pesos de la capa oculta es:

$$w = w + learningParameter * \delta * z$$

Y la actualización de la matriz de pesos entre la capa de entrada y la capa oculta es:

$$V = V + learningParameter * \Delta * x$$

Siguiendo este algoritmo los resultados obtenidos no son iguales a los que obtiene ScikitLearn, esto se debe a que ScikitLearn realiza una optimización cuando los problemas son de tipo binario *[5]*.

Además de este algoritmo en cada época hay que calcular el Error Cuadrático Medio (ECM de ahora en adelante) pero dado que esto únicamente es necesario para la condición de detección de convergencia se explica en el correspondiente apartado.

4.2.2 Entrenamiento mediante retropropagación Batch

El algoritmo es similar al Online, únicamente hay que tener las siguientes consideraciones:

- No se permutan los datos entre épocas.
- No se aplica la actualización de pesos entre cada vector de datos, sino que se van acumulando dichas actualizaciones (valores de δ y Δ_i) y se aplican de forma conjunta al finalizar la época.

4.2.3 Clasificación tras el entrenamiento

Clasificar un conjunto de valores de entrada una vez la red está entrenada es relativamente sencillo. Simplemente calculamos, al igual que hacemos en el entrenamiento, el valor de la neurona de salida y en función del mismo clasificamos dicho conjunto de valores de entrada.

Esto es, dado un conjunto de valores de entrada x:

$$z = sigmoid(V^T * x)$$

$$Y = sigmoid(z * w)$$

$$Si\ Y < 0.5: Clasificamos\ como\ Clase\ 0$$
$$Si\ Y \geq 0.5: Clasificamos\ como\ Clase\ 1$$

Nótese que en el cálculo del valor de la salida de la red neuronal, esto es Y, no se añade ninguna clase de ruido a diferencia del que añadimos en la fase de entrenamiento.
Como se explica en el *Apartado 6.2 Trabajo futuro* podría añadirse una mejora a la clasificación y añadir un parámetro que desplazara el umbral de clasificación y este no sea por lo tanto 0.5 de manera fija.

La clasificación quedaría por lo tanto, siendo λdicho parámetro con un valor en el intervalo [-0.5, 0.5].

$$Si\ Y < 0.5 +/- \lambda: Clasificamos\ como\ Clase\ 0$$
$$Si\ Y \geq 0.5 +/- \lambda: Clasificamos\ como\ Clase\ 1$$

4.2.4 Early stopping

El *Early Stopping* nos permite detener el entrenamiento cuando la red neuronal ha convergido y ya no es necesario seguir entrenando, es más, seguir entrenando puede ocasionar posteriores peores resultados a la hora de clasificar.

Por defecto, el *early stopping* no está activado, y se activa cuando el parámetro *tolerance* en el constructor recibe algún valor.

Para explicar la lógica implementada es necesario conocer la formulación del error cuadrático medio (ECM), tomando como $\hat{x}$ el vector de valores predecidos y x el vector de valores esperados:

$$ECM(x, \hat{x}) = \frac{\Sigma \frac{1}{n} (\hat{x} - x)^2}{n}$$

Si al calcular el ECM de los valores esperados y predecidos de salida de la red neuronal en una época, este es mayor que el de la época anterior, y mayor que el de la época anterior a la anterior más la tolerancia, se detecta que hemos llegado al límite de convergencia y por lo tanto activamos el *early stopping*, asignando los pesos de la red neuronal a los pesos que tenía 2 épocas atrás y finalizando la fase de entrenamiento.

Por lo tanto activamos el *early stopping*, tomando $t$como expresión del tiempo en épocas, si:

$$ECM(t) > ECM(t-1) \; y \; ECM(t) > ECM(t-2) + tolerance$$

La razón de comparar 3 épocas y no 2 es la de disminuir los falsos positivos.

4.2.5 Métodos

Se han creado tres métodos asociados a la clase NoiseNeuralNetwork. Se marcan con asterisco * los parámetros opcionales:

El constructor:

> NoiseNeuralNetwork(hiddenLayerSize, learningParameter, noiseScale*, noiseType*, tolerance*).
>
> - hiddenLayerSize: Número de neuronas (sin incluir la neurona bias) de la capa oculta.
>
> - learningParameter: Cantidad de desplazamiento en la actualización de los pesos tanto de la matriz entre la capa de entrada y la capa oculta, como del vector entre la capa oculta y la capa de salida.
>
> - noiseScale (Opcional): Escala del ruido generado. Por defecto es 0, es decir, el entrenamiento no se verá afectado por ningún ruido.

- noiseType (Opcional): Tipo de ruido, tiene dos posible valores.Por defecto es ‘normal’.
    - ‘normal’: El ruido seguirá una distribución normal estándar, es decir, con media en 0 y una desviación típica de 1, con un dominio entre -∞ y +∞.
    - ‘uniform’: El ruido seguirá una distribución uniforme en el rango de valores [-1 , 1].

- tolerance (Opcional): Factor de tolerancia para el *early stopping*. Por defecto es None, si se le asigna cualquier valor (Incluido 0) activará la comprobación del *early stopping* en cada época.

- Devuelve la instancia de la clase, esto es la implementación del MLP propuesto.

Un método para el entrenamiento de la red:

NoiseNeuralNetwork.train(data, classes, epochNumber*, useBatch*, init*)

- data: Conjunto de ejemplos sobre el que se va a entrenar la red neuronal, es altamente recomendable, sino imprescindible que estos datos estén normalizados. No debe incluir los valores de las clases de cada vector de atributos.

- classes: Vector con las clases, 0 o 1, a la que pertenece cada vector de atributos de data.

- epochNumber (Opcional): Número máximo de épocas para la fase de entrenamiento, puede ser interrumpido por el *early stopping* si este está activado. Por defecto es 200.

- useBatch (Opcional): Indica si se debe entrenar utilizando retropropagación *Online* o *Batch*, si es *True* utiliza *Batch*. Por defecto *False*.

- init (Opcional): Indica si deben inicializarse los pesos antes de entrenar.

- No devuelve ningún valor.

Un método para la predicción de un vector de atributos:

NoiseNeuralNetwork.predict(attributes)

- attributes: Vector de atributos sobre el que se desea predecir la clase, la red ha debido ser entrenada previamente para que el resultado sea fiable.

- Devuelve 0 si el vector de atributos attributes pertenece a la Clase 0 y 1 si pertenece a la Clase 1.

4.2.6 Ejemplo

Un ejemplo de la utilización de todos estos métodos:

```
#Leemos dataset desde el archivo separado por comas
csv = np.genfromtxt('example.csv', delimiter=',')
#Seleccionamos nuestro conjunto de datos, es decir, todas las columnas salvo
#la correspondiente a la clase
data = csv[:, 0:-1]
#Normalizamos el conjunto de datos
data = preprocessing.scale(data)
#Convertimos la columna de clases leidas del fichero en un vector columna
classes = csv[:, -1].reshape(csv.shape[0],1)    # select last column

HIDDEN_SIZE = 11
LEARNING_PARAMETER = 0.1

#Creamos la red neuronal con los parametros que deseamos
n = NoiseNeuralNetwork(HIDDEN_SIZE,LEARNING_PARAMETER,
                       noiseScaleParameter=0.5,
                       tolerance=0.0001)
#La entrenamos
n.train(data, classes, 100)
#Predecimos el valor que devuelve la red neuronal para (0, 1)
print("Clase de (0, 1): " + str(n.predict([0,1])))
```

Figura 7. Ejemplo de código Python utilizando las funciones implementadas en la clase NoiseNeuralNetwork.

5 Pruebas y resultados

5.1 Datasets

Todos los datasets utilizados para realizar el estudio y las pruebas son binarios, es decir, consisten en clasificar los datos en 2 clases.

5.1.1 Datasets creados

Se ha generado un único dataset y es el primero sobre el que se han realizado las pruebas.

Se resume su contenido aunque ya haya sido explicado en el *Apartado 3.2.2*.

- Gaussianas: Consta de 500 ejemplos, en este caso puntos en el plano, pertenecientes a dos clases completamente equilibradas, esto es, 250 ejemplos de cada clase. Con dos atributos que son las coordenadas X e Y que representan cada punto. No hay ruido de clases ni faltan atributos. Todos los datos son numéricos y continuos.

5.1.2 Datasets de problemas reales

Se han utilizado tres datasets de problemas de clasificación reales *[8]*. Estos han sido tratados por trabajos de investigación y papers de otros autores:

- Pima: Contextualizado en el estudio de la diabetes. Consta de 768 ejemplos con algo de ruido en las clases. No faltan atributos. Todos los atributos son numéricos.

- German: Contextualizado en la clasificación de clientes según su estabilidad/riesgo financiero. Consta de 100 ejemplos. No faltan atributos. Aunque el dataset original contiene atributos numéricos y categóricos, se ha utilizado una versión en la que todos los datos son numéricos.

- Breast: Contextualizado en el estudio del cáncer de mama. Consta de 699 ejemplos. Faltan atributos. Todos los atributos son numéricos.

5.2 Pruebas de implementación

5.2.1 Debugging

Es la primera prueba de implementación que se realizó, comprobación de todos los valores del MLP y la actualización de sus pesos durante 1 época mediante *debugging*.

5.2.2 Pruebas de resultados

Comprobación de resultados sin aplicar ruido en el dataset gaussianas. Utilizando 66% del dataset como entrenamiento y 33% como test.

Tras ejecutarlo 100 veces el porcentaje de acierto contra train es: 95.03±0.86
Tras ejecutarlo 100 veces el porcentaje de acierto contra test es: 90.79±0.85

5.3 Estudio de la afección del ruido a la clasificación

5.3.1 Procedimiento de pruebas

La validación cruzada estratificada tiene un gran peso en este procedimiento de pruebas. Consiste en dividir el conjunto de datos con el que se va a entrenar en N (5 en este estudio) partes de un tamaño similar, al ser estratificada, significa que en todas las partes hay un número similar de individuos de cada clase.

De estas N partes o iteraciones, N-1 partes se utilizan para entrenar la red neuronal (en el contexto de este TFG) y la parte restante se utiliza para validar el entrenamiento realizado. Este proceso se repite N veces alternando la parte de test en cada iteración. La puntuación total de la validación cruzada es la media de las puntuaciones de entrenar y validar en las N iteraciones.

La utilidad de la validación cruzada es que asegura que los resultados obtenidos son independientes de las particiones train y test seleccionadas, estabilizando por tanto los resultados obtenidos.

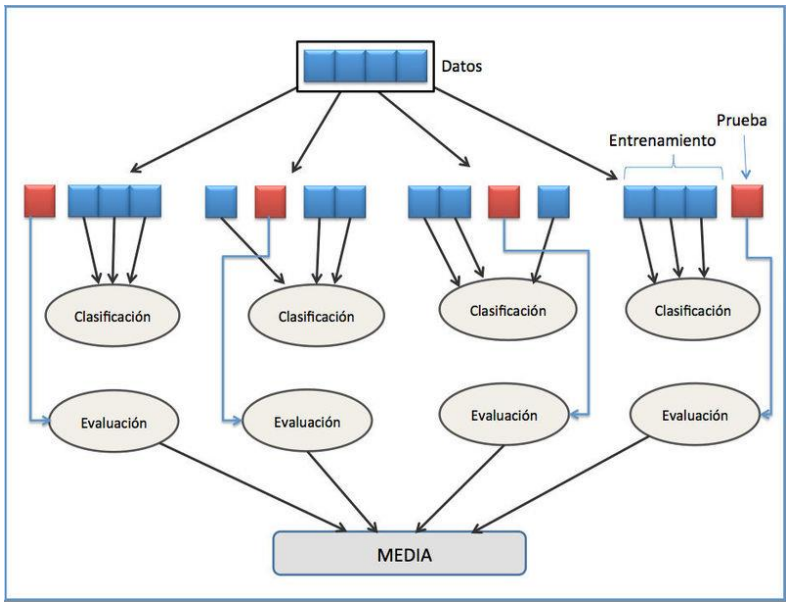

Figura 8. Esquema del procedimiento de validación cruzada con 4 iteraciones. Fuente: Wikipedia.

El procedimiento de pruebas seguido para realizar este estudio ha sido: Para cada dataset se ha realizado el siguiente procedimiento:

1. Normalización del dataset.
2. División del dataset en dos particiones, una partición de test 25 % y otra de train.
    a. Creación de las particiones para la validación cruzada estratificada.
    b. Búsqueda conjunta de los parámetros HiddenLayerSize y LearningParameter que maximizan la probabilidad de acierto. Se ha utilizado una malla formada por distintos valores de estos parámetros, de forma que cada par de estos recibiera una puntuación.
        i. El score para un par (HiddenLayerSize , LearningParameter) se ha definido como la media de las puntuaciones obtenidas al realizar una validación cruzada de cinco pliegues sobre la partición de train.
        ii. Además de calcular el score se ha calculado la desviación típica de cada par para poder realizar una tabla con una visión global de toda la malla.
    c. Tras encontrar dicho par, se comprueba el resultado de entrenar con toda la partición de train y testearlo contra la partición de test y de train por separado. Se hace N veces, también se obtiene la media y la desviación típica. En las pruebas realizadas para este estudio se ha tomado N = 20.
    d. Fijando como parámetros el par ya calculado y comprobado, se realiza una búsqueda similar para encontrar el valor óptimo de la escala del ruido. También se calcula la media y la desviación típica de cada valor de escala probado.
    e. Al igual que con el par, se comprueba el resultado obtenido contra la partición de test y de train por separado, ejecutándolo N veces y obteniendo la media y desviación típica.
    f. Como resultado obtenemos una comparación del entrenamiento de la red con y sin ruido, además de tres parámetros que maximizan (dentro de la búsqueda realizada) la probabilidad de acierto.

Es importante recalcar que la partición de test únicamente se utiliza para comprobar los resultados obtenidos.

Los valores estudiados son:

- HiddenLayerSize: 5, 7, 10, 15, 20.
- LearningParameter: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0.
- NoiseScale: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30

A continuación se muestra un esquema representativo de este proceso:

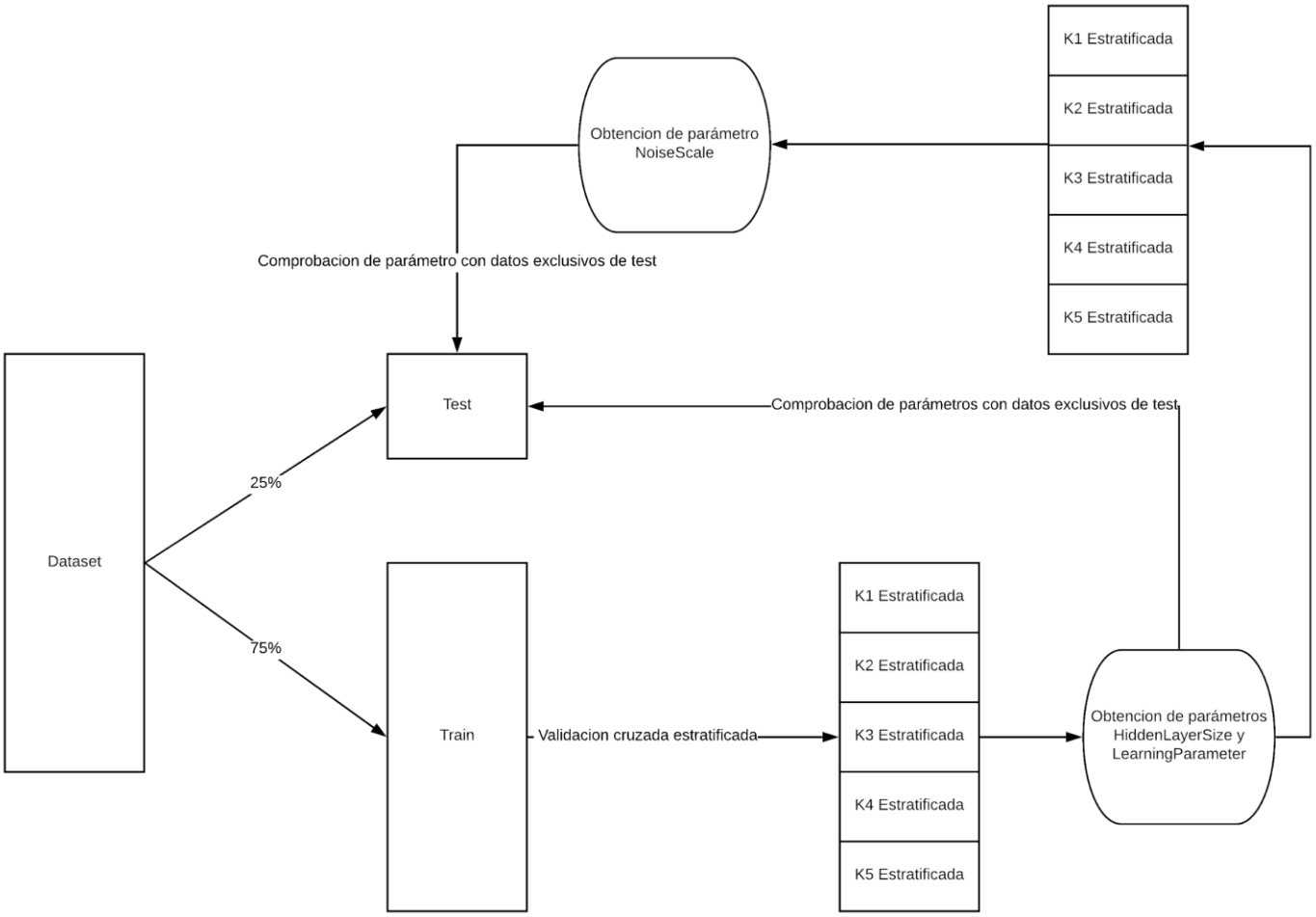

Figura 9. Esquema del proceso de pruebas realizado para cada dataset.

5.3.2 Resultados

Pueden encontrarse en el Apéndice las tablas para la optimización de todos los parámetros para los distintos datasets.

Todos los resultados en las tablas están redondeados a dos decimales.

Tras optimizar los tres parámetros (Número de neuronas en la capa oculta, constante de aprendizaje y escala del ruido) mediante el procedimiento explicado en el apartado anterior, se hace un estudio sobre las particiones de test y train con los distintos modelos de ruido: Ninguno, normal y uniforme.

Por lo general las estimaciones de la precisión obtenidas utilizando el método de entrenamiento retropropagación *Batch* son inferiores a la utilización de la variante *Online*, por este motivo todos los resultados presentados a continuación han sido obtenidos entrenando con retropropagación *Online.*

Resultados globales contra las particiones de test:

Test	**Sin ruido**	**Ruido normal**	**Ruido uniforme**
Gaussianas	94.67±0.79	94.59±0.90	94.97±0.96
Pima	76.72±1.61	75.98±1.38	76.53±1.20
German	74.47±0.79	74.68±0.89	74.04±0.94
Breast	94.96±1.10	95.30±0.70	95.19±0.70

Tabla 1. Estimación de la precisión contra la partición de test para todos los datasets, métodos de inyección de ruido con los mejores parámetros calculados.

Resultados globales contra las particiones de train:

Train	**Sin ruido**	**Ruido normal**	**Ruido uniforme**
Gaussianas	94.20±0.43	93.73±0.80	94.18±0.68
Pima	78.29±1.09	77.77±1.14	78.35±1.03
German	79.27±0.82	79.31±0.70	79.36±0.60
Breast	96.12±0.48	96.13±0.52	96.32±0.45

Tabla 2. Estimación de la precisión contra la partición de train para todos los datasets, métodos de inyección de ruido con los mejores parámetros calculados.

A continuación se estudia el comportamiento de la precisión en función del número de épocas de entrenamiento para cada dataset utilizando los parámetros óptimos ya calculados y entrenamiento con retropropagación *Online*:

En el dataset gaussianas:

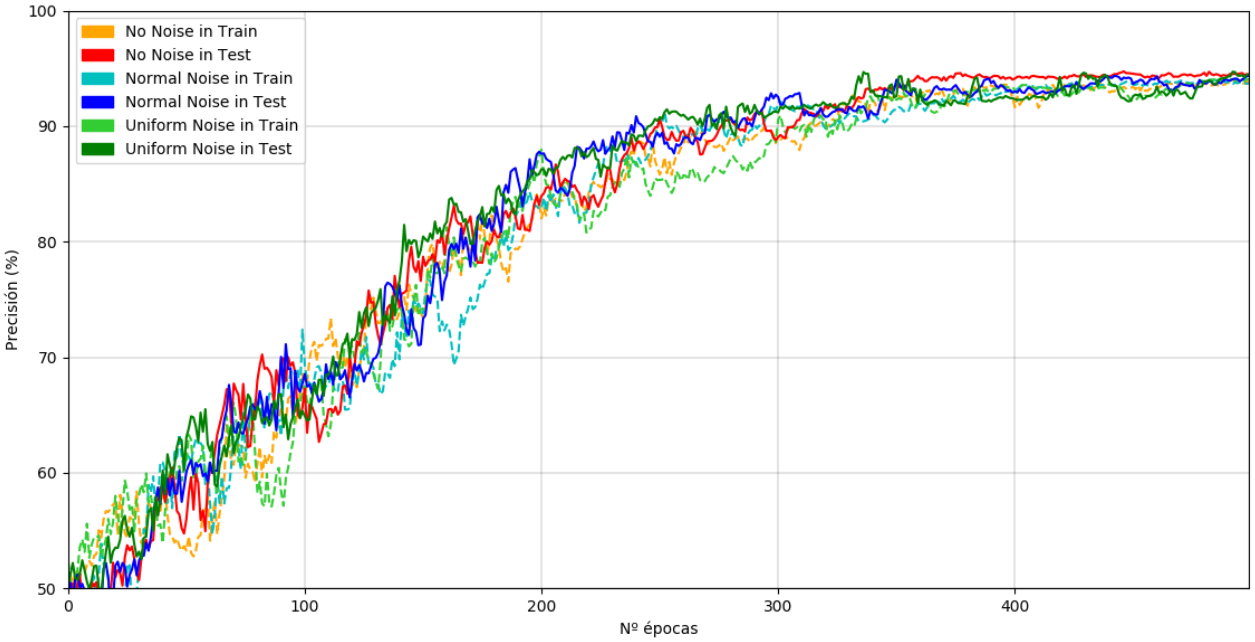

Figura 10. Estimación de la precisión en el dataset gaussianas para todos los modelos de ruido sobre ambas particiones en función del número de épocas de entrenamiento.

En el dataset Pima:

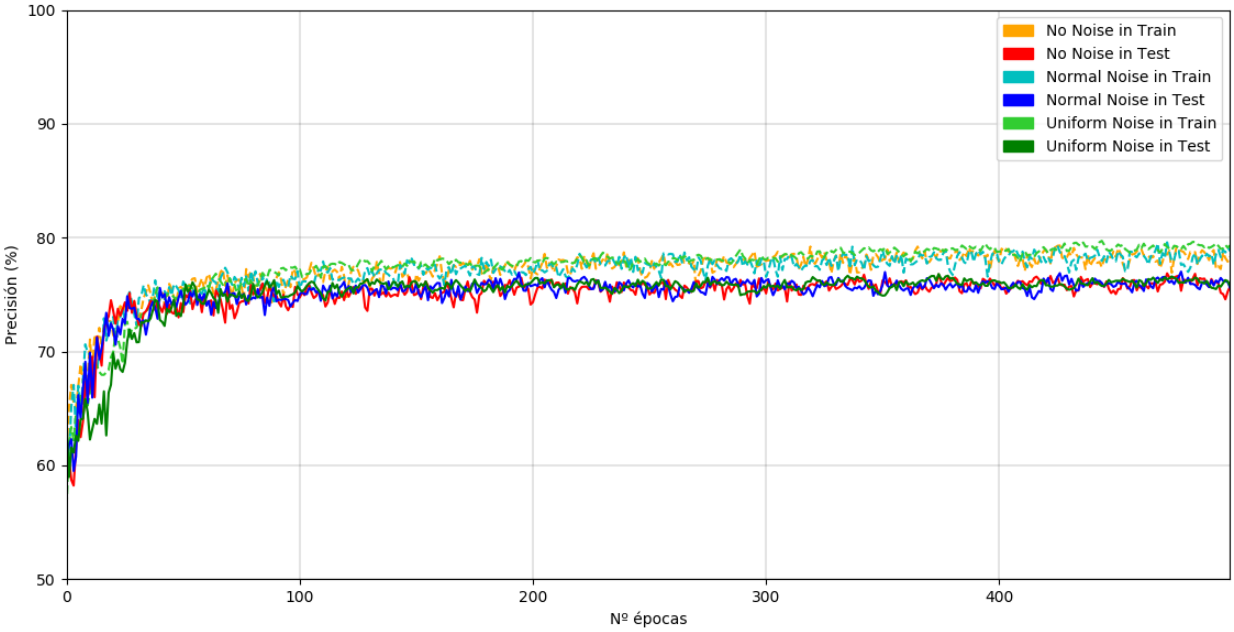

Figura 11. Estimación de la precisión en el dataset Pima para todos los modelos de ruido sobre ambas particiones en función del número de épocas de entrenamiento.

En el dataset German:

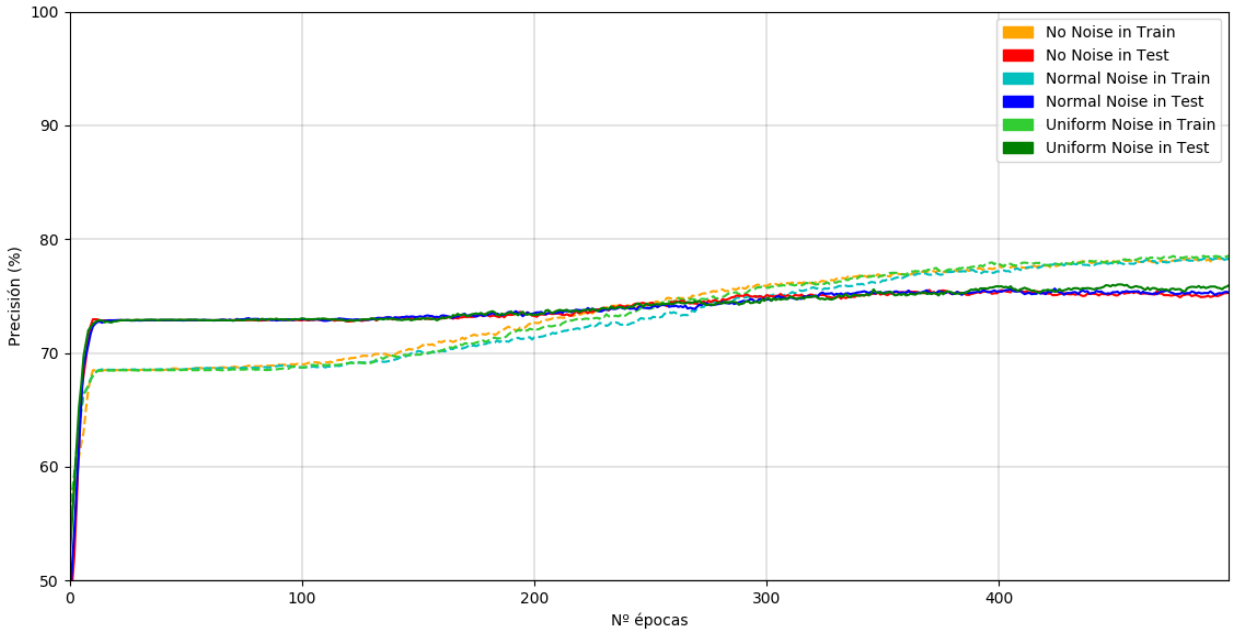

Figura 12. Estimación de la precisión en el dataset German para todos los modelos de ruido sobre ambas particiones en función del número de épocas de entrenamiento.

En el dataset Breast:

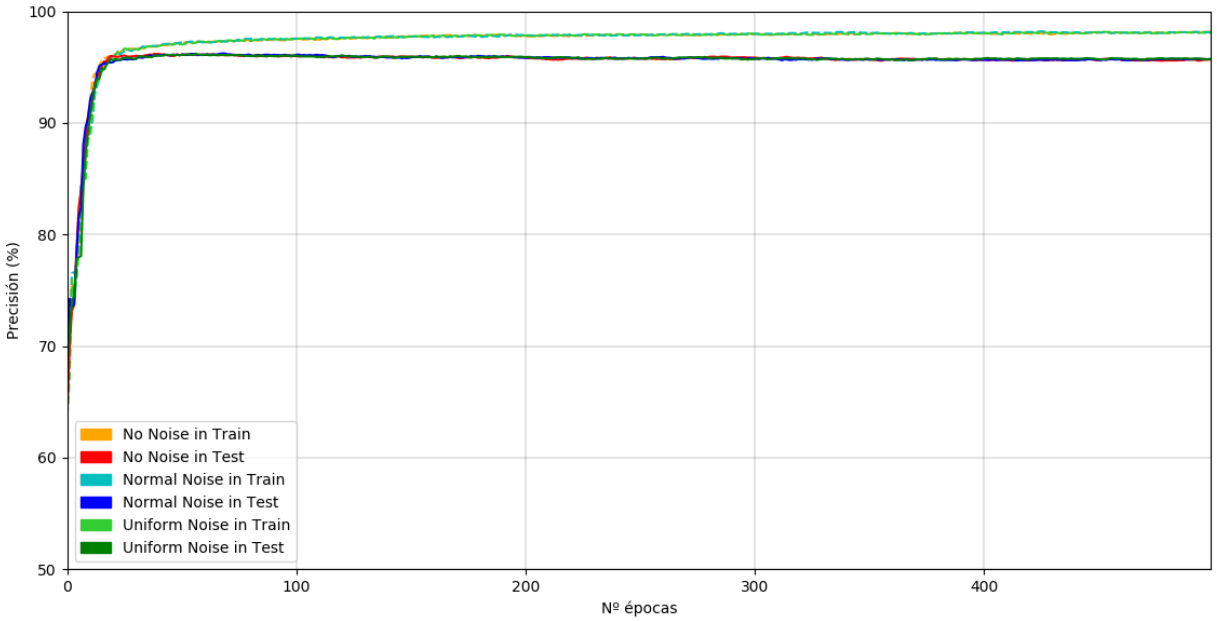

Figura 13. Estimación de la precisión en el dataset Breast para todos los modelos de ruido sobre ambas particiones en función del número de épocas de entrenamiento.

Las líneas discontinuas y de colores más claros representan la precisión en train mientras que las continuas y con colores más oscuros representan la precisión en test, tal y como se indica en las leyendas de las distintas figuras.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Con la realización de este TFG se ha aprendido y profundizado en el funcionamiento de los perceptrones multicapa, los problemas que sufren normalmente y los distintos enfoques que se suelen aplicar sobre dichos problemas.

Antes de realizar la implementación del MLP se realizó una investigación de librerías externas ya existentes de aprendizaje automático y clasificación, esto permitió conocer más en profundidad la implementación y optimizaciones que hace ScikitLearn por debajo en los MLP, y familiarizarse con el funcionamiento y uso de Keras que permite diseñar redes neuronales por capas.

A nivel de resultados y como se muestra en la siguiente tabla, la escala del ruido es importante ya que distintos datasets necesitan distintas escalas del ruido para no distorsionar el aprendizaje:

	Gaussianas	**Pima**	**German**	**Breast**
Normal	1	0.1	0.01	0.03
Uniforme	0.03	0.003	0.003	0.001

Tabla 3. Escala del ruido óptima en cada dataset utilizando como método de entrenamiento retropropagación Online.

En la *Tabla 1* de resultados no podemos apreciar un patrón claro y determinante de si el ruido mejora o empeora la clasificación.

Por lo tanto en vista de las *Figuras 10-13* la inyección de ruido en la etapa de entrenamiento de la red neuronal no parece mejorar la precisión de la clasificación obtenida en comparación con el obtenido sin la inyección del mismo.

Tampoco permite alcanzar el punto de convergencia en un menor número de épocas, lo cual se apreciaría si la curva de aprendizaje tuviera una mayor inclinación o pendiente.

A continuación se muestra un ejemplo donde se puede ver la *Figura 11* ampliada en torno a las épocas en las que converge, esto es, donde la precisión sobre la partición de test deja de mejorar. Se puede observar que esta convergencia es alcanzada antes por la red neuronal entrenada sin la inyección de ruido, que por las otras dos.

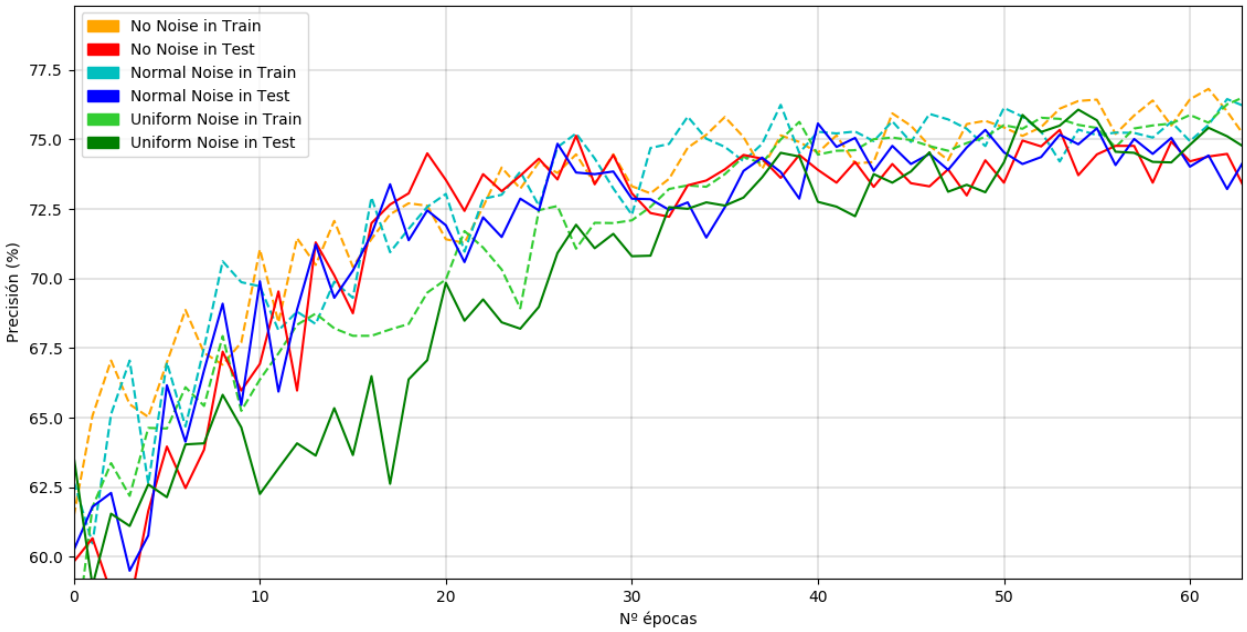

Figura 14. Ampliación de la Figura 11 en la zona de convergencia.

Como se puede apreciar la línea roja continua, que representa la estimación de la precisión sobre la partición de test, es la primera en converger, es decir aproximarse a la horizontalidad (sin tener en cuenta el ruido).

De hecho, la pendiente de la línea verde oscuro continúa, que representa la estimación de la precisión sobre la partición de test inyectando ruido uniforme, es menor y supone un retraso (respecto a la no inyección de ruido) en alcanzar la convergencia cercano a 20 épocas.

Concluyendo, aunque la inyección de ruido propuesta en este TFG no representa un gran aumento de coste computacional a la hora de realizar el entrenamiento, no hay motivos determinantes por la que resulte beneficiosa para el entrenamiento, y como se ha explicado anteriormente, sí que puede llegar a ser levemente perjudicial.

6.2 Trabajo futuro

Algunos puntos sobre los que se pueden continuar este TFG son los siguientes:

- Ampliación del diseño para funcionar en problemas con más de dos clases de salida, es decir, que sean multiclase y no binarios. Además realizar el mismo tipo de pruebas con estos problemas para ver cómo se comportan con estos.

- Ampliación del diseño para soportar más de una capa oculta y permitir la adición opcional de una neurona de ruido en cada una de las capas ocultas.
    - Consecuentemente permitir distintas escalas de ruido para cada neurona.
    - Con el fin de no introducir cada escala a mano, y la dificultad de saber que escala sería necesaria en cada capa, puede ser interesante buscar algún método de ajuste automático para la escala de este ruido en cada neurona.

- Añadir otro parámetro a la red neuronal, desplazamiento en la predicción que permita definir un desplazamiento del umbral que separa la clasificación en ambas clases.

- Generar un método que permita calcular automáticamente el mejor valor de desplazamiento en la predicción.

- Utilizar un método más preciso/exhaustivo para la optimización de parámetros, por ejemplo, algoritmos genéticos.

Referencias

Ordenadas por orden cronológico:

[1] Marvin Minsky and Seymour Papert, “Perceptrons. An Introduction to Computational Geometry”, M.I.T. Press, Cambridge, Mass., 1969.

[2] Yves Grandvalet, Stéphane Canu and Stéphane Boucheron, “Noise Injection: Theoretical Prospects”, Neural Computation, Volume 9, Pages 1093-1108, 1997.

[3] Chuan Wang and J. C. Principe, “Training neural networks with additive noise in the desired signal”, Trans. Neur. Netw, Volume 10, Pages 1511-1517, 1999.
URL https://doi.org/10.1109/72.809097

[4] Christopher M. Bishop, “Pattern Recognition and Machine Learning (Information Science and Statistics)”, Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E ,“Scikit-learn: Machine Learning in Python”}, Journal of Machine Learning Research, Volume 12, Pages 2825-2830, 2011.
URL http://scikit-learn.org/stable/modules/neural_networks_supervised.html

[6] Chollet, François and others.”Keras”, 2015.
URL https://keras.io

[7] Ian Goodfellow and Yoshua Bengio and Aaron Courville, “Deep Learning”, M.I.T. Press, 2016.

[8] Dua, D. and Karra Taniskidou, Efi, “UCI Machine Learning Repository“, Irvine, CA: University of California, School of Information and Computer Science, 2017.
URL http://archive.ics.uci.edu/ml

[9] Noh H., You T., Mun J., and Han B. “Regularizing Deep Neural Networks by Noise: Its Interpretation and Optimization”, ArXiv e-prints, 2017.

[10] Marcelino Lázaro, Monson H. Hayes, Aníbal R. Figueiras-Vidal, “Training neural network classifiers through Bayes risk minimization applying unidimensional Parzen windows”, Pattern Recognition, Volume 77, Pages 204-215, 2018.
URL https://doi.org/10.1016/j.patcog.2017.12.018

Glosario

Ordenadas por orden alfabético:

Dataset	Conjunto de datos
ECM	Error Cuadrático Medio
MLP	MultiLayer Perceptron
RNA	Red neuronal artificial
TFG	Trabajo de Fin de Grado

Anexos

A Tablas de resultados

En primer lugar se muestran los resultados de la búsqueda de los parámetros óptimos de la constante de aprendizaje y el número de neuronas en la capa oculta. Se muestra en negrita el mejor resultado en función de la estimación de su precisión y los parámetros seleccionados.

Gaussianas	5	7	**10**	15	20
0.001	*57.17±13.25*	*50.61±1.11*	*48.48±1.59*	*47.27±7.02*	*48.12±16.77*
0.003	*45.15±20.74*	*54.3±8.4*	*62.9±17.0*	*57.04±9.85*	*77.04±15.18*
0.01	*66.92±14.07*	*72.06±12.61*	*73.55±13.46*	*79.97±10.45*	*86.33±5.04*
0.03	*80.32±11.13*	*89.65±6.51*	*91.25±4.33*	*92.71±2.97*	*92.73±2.6*
0.1	*93.34±3.78*	*94.85±2.05*	***94.86±2.25***	*93.33±2.06*	*94.24±1.78*
0.3	*94.55±2.64*	*93.33±1.55*	*94.24±3.24*	*94.55±2.05*	*94.85±2.81*
1	*94.55±2.64*	*92.75±2.91*	*93.65±2.41*	*93.93±2.54*	*93.94±1.65*

Tabla 4. Resultados de la búsqueda en rejilla en el dataset gaussianas.

Pima	5	7	10	**15**	20
0.001	*60.06±13.70*	*67.0±0.69*	*59.87±13.6*	*69.17±1.57*	*67.59±1.04*
0.003	*67.19±0.33*	*67.0±0.30*	*67.0±0.30*	*67.19±0.33*	*67.19±0.94*
0.01	*67.2±0.61*	*67.0±0.30*	*68.58±2.99*	*67.8±1.40*	*67.79±2.53*
0.03	*72.14±4.30*	*72.52±5.44*	*73.9±3.12*	*74.7±1.77*	*76.28±4.32*
0.1	*76.07±3.04*	*75.87±3.14*	*75.87±3.77*	*76.06±4.19*	*75.08±4.75*
0.3	*76.65±4.94*	*75.86±4.28*	*77.04±6.13*	***77.83±4.75***	*76.85±4.34*
1	*76.65±4.29*	*75.86±5.52*	*77.25±5.05*	*75.67±3.93*	*75.87±4.81*

Tabla 5. Resultados de la búsqueda en rejilla en el dataset Pima.

German	**5**	7	10	15	20
0.001	*69.85±0.21*	*69.85±0.21*	*69.85±0.21*	*69.55±0.95*	*69.85±0.21*
0.003	*69.85±0.21*	*70.0±0.30*	*69.55±0.71*	*71.07±1.28*	*69.85±1.71*
0.01	*71.21±2.29*	*71.37±1.66*	*72.72±2.95*	*74.39±2.61*	*74.39±2.76*
0.03	***78.95±2.31***	*77.42±2.8*	*77.73±1.70*	*77.73±2.06*	*76.82±2.50*
0.1	*77.12±1.92*	*77.13±2.33*	*77.58±1.94*	*78.33±3.21*	*78.03±1.99*
0.3	*76.97±1.75*	*76.67±2.46*	*76.21±1.16*	*75.91±2.48*	*75.45±2.90*
1	*73.94±1.28*	*74.99±3.53*	*74.55±2.96*	*74.7±2.67*	*74.7±3.82*

Tabla 6. Resultados de la búsqueda en rejilla en el dataset German.

Breast	5	7	**10**	15	20
0.001	*64.25±14.66*	*72.9±9.77*	*69.43±12.03*	*82.85±10.61*	*76.8±5.40*
0.003	*72.85±11.41*	*83.49±8.42*	*85.27±8.34*	*90.88±3.42*	*92.62±2.12*
0.01	*92.83±6.36*	*94.79±2.43*	***97.18±2.34***	*96.31±1.31*	*96.96±2.22*
0.03	*96.96±2.22*	*96.74±1.95*	*96.74±2.57*	*96.96±2.22*	*96.96±2.22*
0.1	*96.96±2.11*	*96.53±2.32*	*96.74±2.57*	*96.53±1.74*	*96.53±1.87*
0.3	*96.96±2.42*	*96.09±2.63*	*96.09±2.63*	*96.74±2.48*	*96.31±2.80*
1	*96.74±2.28*	*97.18±2.44*	*96.31±2.8*	*96.31±2.72*	*96.53±2.11*

Tabla 7. Resultados de la búsqueda en rejilla en el dataset Breast.

Con estos dos primeros parámetros fijados se estudia la escala del ruido (de ambos tipos) en cada dataset, donde se marca en negrita (por columnas) la mejor escala de cada tipo de ruido:

Gaussianas	Normal	Uniforme
0.001	*93.34±2.95*	*93.64±2.40*
0.003	*93.93±2.15*	*93.95±2.11*
0.01	*93.65±2.90*	*94.24±2.42*
0.03	*93.63±2.24*	***94.55±1.82***
0.1	*93.63±2.24*	*93.02±2.83*
0.3	*93.04±2.43*	*93.01±3.02*
1	***94.85±2.05***	*93.63±3.10*
3	*89.06±4.22*	*92.73±2.42*
10	*59.61±11.14*	*63.01±15.82*
30	*50.3±1.23*	*57.03±11.64*

Tabla 8. Resultados búsqueda de escalas del ruido en el dataset gaussianas.

Pima	Normal	Uniforme
0.001	*76.26±5.10*	*75.46±4.68*
0.003	*75.66±4.86*	***78.03±5.08***
0.01	*73.5±3.37*	*74.29±3.64*
0.03	*75.27±5.01*	*75.66±5.22*
0.1	***77.04±4.58***	*75.27±4.88*
0.3	*77.04±5.28*	*76.65±5.42*
1	*76.46±4.72*	*75.28±4.88*
3	*76.06±3.86*	*76.66±3.31*
10	*67.0±0.30*	*66.8±0.50*
30	*67.0±0.30*	*67.0±0.30*

Tabla 9. Resultados búsqueda de escalas del ruido en el dataset Pima.

German	Normal	Uniforme
0.001	*76.97±1.99*	*77.12±1.96*
0.003	*76.06±3.03*	***78.49±1.74***
0.01	***78.48±2.65***	*77.11±1.96*
0.03	*78.03±2.74*	*76.52±1.05*
0.1	*77.28±2.56*	*77.57±2.25*
0.3	*76.97±1.98*	*75.91±2.36*
1	*76.37±3.07*	*76.06±2.99*
3	*69.85±0.21*	*71.52±3.26*
10	*69.85±0.21*	*69.85±0.21*
30	*69.85±0.21*	*69.85±0.21*

Tabla 10. Resultados búsqueda de escalas del ruido en el dataset German.

Breast	Normal	Uniforme
0.001	*96.74±1.95*	***96.96±1.44***
0.003	*95.87±1.45*	*96.53±1.74*
0.01	*96.74±2.57*	*96.31±1.48*
0.03	***96.96±1.87***	*96.53±1.27*
0.1	*95.44±1.45*	*95.88±1.87*
0.3	*95.44±2.52*	*96.09±1.77*
1	*95.88±1.87*	*96.53±2.11*
3	*72.69±11.97*	*92.62±5.26*
10	*66.35±6.61*	*69.64±7.95*
30	*45.39±17.08*	*65.27±5.55*

Tabla 11. Resultados búsqueda de escalas del ruido en el dataset Breast.